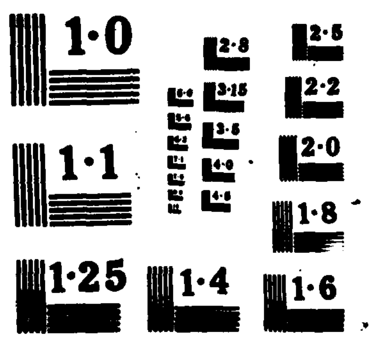
AD-A169 938 YIELD AND PERFORMANCE ENHANCEMENT THROUGH REDUNDANCY IN VLSI AND WSI MULT..(U) MASSACHUSETTS UNIV AMHERST DEPT OF ELECTRICAL AND COMPUTER EN.. I KOREN ET AL. 1986 1/1

UNCLASSIFIED AFOSR-TR-86-0389 AFOSR-84-0052 F/G 9/2 NL


1·0
2·8
2·5
3·15
2·2
3·5
1·1
2·0
4·0
4·5
1·8
1·25
1·4
1·6

AD-A169 930

(2)

REPORT DOCUMENTATION PAGE

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)
AFOSR-TR- 86-0389

6a. NAME OF PERFORMING ORGANIZATION
University of Massachusetts at Amherst

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
Air Force Office of Scientific Research

6c. ADDRESS *(City, State and ZIP Code)*
Dept. of Electrical & Computer Engineering
School of Engineering
Amherst, MA 01003

7b. ADDRESS *(City, State and ZIP Code)*
Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332-6448

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
AFOSR

8b. OFFICE SYMBOL *(If applicable)*
NM

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
AFOSR-84-0052

8c. ADDRESS *(City, State and ZIP Code)*
Bolling AFB DC 20332-6448

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	A2	

11. TITLE *(Include Security Classification)*
"YIELD AND PERFORMANCE ENHANCEMENT THROUGH REDUNDANCY IN VLSI AND WSI MULTI-PRECESSOR SYSTEMS"

12. PERSONAL AUTHOR(S)
Dr.'s Israel Koren and Dhiraj K. Pradhan

13a. TYPE OF REPORT
Technical

13b. TIME COVERED
FROM ______ TO ______

14. DATE OF REPORT *(Yr., Mo., Day)*
1986

15. PAGE COUNT
35 + diagrams

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

DTIC FILE COPY

New challenges have been brought to fault-tolerant computing and processor architecture research because of developments in IC technology. One emergent area is development of architectures, built by interconnecting a large number of processing elements on a single chip or wafer. Two important areas, related to such VLSI processor arrays, are the focus of this paper; they are fault-tolerance, and yield improvement techniques.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Captain John P. Thomas

22b. TELEPHONE NUMBER *(Include Area Code)*
(202) 767- 5026

22c. OFFICE SYMBOL
NM

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

#7

Fault-tolerance in these VLSI processor arrays is of real practical significance; it provides for much-needed reliability improvement. Therefore, we first describe the underlying concepts of fault-tolerance at work in these multi-processor systems. These precepts are useful to then present certain techniques that will incorporate fault-tolerance integrally into the design. In the second part of the paper we discuss models that evaluate how yield enhancement and reliability improvement may be achieved by certain fault-tolerant techniques.

YIELD AND PERFORMANCE ENHANCEMENT THROUGH REDUNDANCY IN VLSI AND WSI MULTI-PROCESSOR SYSTEMS +

Israel Koren * and Dhiraj K. Pradhan **

ABSTRACT

New challenges have been brought to fault-tolerant computing and processor architecture research because of developments in IC technology. One emergent area is development of architectures, built by interconnecting a large number of processing elements on a single chip or wafer. Two important areas, related to such VLSI processor arrays, are the focus of this paper; they are fault-tolerance, and yield improvement techniques.

Fault-tolerance in these VLSI processor arrays is of real practical significance; it provides for much-needed reliability improvement. Therefore, we first describe the underlying concepts of fault-tolerance at work in these multi-processor systems. These precepts are useful to then present certain techniques that will incorporate fault-tolerance integrally into the design. In the second part of the paper we discuss models that evaluate how yield enhancement and reliability improvement may be achieved by certain fault-tolerant techniques.

Index Terms - Yield enhancement, reliability, computational availability, processor arrays, fault-tolerance, redundancy, diagnosis, reconfiguration.

+ This work was supported in part by AFOSR contract 84-0052.

* I. Koren is with the Departments of Electrical Engineering and Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel.

** D. K. Pradhan is with the Dept. of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003.

I. INTRODUCTION

The evolution of fifth generation computers [44] makes it clear that traditional sequential computer architecture will soon see a striking departure, overtaken by newer architectures which use multiple processors as the state-of-the-art. This particular thrust is enhanced by developments in IC technology [30], creating a widening gap between the technological advances and the architectural capabilities that can exploit these fully.

As a result, much recent research has focused on these new architectural innovations, especially those created by interconnecting multiple processing elements (PEs). One important class of such architectures is VLSI systems that interconnect a very large number of simple processing cells, all on a single chip or wafer. Concerns about fault-tolerance in VLSI-based systems stem from the two key factors of reliability and yield enhancements. Low yield is a problem of increasing significance as circuit density grows. One solution suggests improvement of the manufacturing and testing processes, to minimize manufacturing faults. However, this approach is not only very costly, but also quite difficult to implement, with the increasing number of components that can be placed on one chip. But incorporating redundancy for fault-tolerance does provide a very practical solution to the low yield problem. This has been demonstrated in practice for high density memory chips and should be extended to other types of VLSI circuits. In general, yield may be enhanced because the circuit can be accepted, in spite of some manufacturing defects, by means of restructuring, as opposed to having to discard the faulty chip. Achieving reliable operation also becomes increasingly difficult with the growing number of interconnected elements and hence, the increased likelihood that faults can occur.

In the design of such fault-tolerant systems, a major architectural consideration becomes the system interconnection. Consequently, one goal of this work is the study of sound fault-tolerant network architectures that can be well-utilized in a wide range of VLSI-based systems. Also, of importance are the related problems of testing, diagnosis, and reconfiguration.

VLSI technology has many promising applications, including the design of special-purpose processors [7], for use as an interconnected array of processing cells on a single chip, as well as the design of super-computers that use wafer-scale technology. These two factors, in conjunction, possess the potential of major innovations in computer architecture.

One principal aspect of such architectures is how fault-tolerance can well be incorporated into such systems. Included here is the problem of the placement of redundant cells so as to achieve the elements of fault-tolerance, yield enhancement, testability and reconfigurability.

II. FAULT-TOLERANCE IN VLSI AND WSI

Two VLSI-based areas in which important innovations are likely to occur are in the wafer-scale integrated architectures, and in the single-chip/multi-processing element architectures. The former has the potential for a major breakthrough with its ability to realize a complete multiprocessing system on a single wafer. This will eliminate the expensive steps required to dice the wafer into individual chips and bond their pads to external pins. In addition, internal connections between chips on the same wafer are more reliable and have a smaller propagation delay than external connections. The latter does make it possible to build a high-speed processor on a single chip, designed by interconnecting a large number of simple PEs. These architectures already have captured the imagination of several computer manufacturers and researchers alike.

As mentioned earlier, the motivation for incorporating fault-tolerance (redundancy) is two fold: yield enhancement and reliability improvement. Both are achieved by restructuring the links so as to isolate the faulty element(s). Various link technologies are available now which allow such restructurability. Included among these are the laser-formed links, MOS links (tristate logic and transistors), fusible links, and so on.

Restructuring capability is either static or dynamic in type. Which type is selected depends on whether restructuring should be performed only once after manufacturing, or an unlimited number of times, as may be required, throughout the operational life.

The issue of fault-tolerance in VLSI and WSI processing arrays has been the subject of recent studies, e.g., [8], [10], [18], [20], [26], [38], [40], [41]. In these publications, various schemes have been proposed that introduce fault-tolerance into the architecture of processor arrays. Because fault-tolerance is an involved subject, completely different schemes might be cost effective in different situations and for different objective functions.

When evaluating a fault-tolerance strategy for multi-processor systems we have to consider the following aspects:

(a) Types of failures to be handled and their probabilities of occurrence.

(b) The costs associated with failure occurrences.

(c) The applicable recovery methods.

(d) The amount of additional hardware needed.

(e) The system objective functions.

Fault-tolerance strategies can be designed to deal with two distinct types of failures, namely, production defects and operational faults. In the current technology, a relatively large number of defects is expected when manufacturing a silicon wafer. Normally, all chips with production flaws are discarded leading to a low yield (expected percentage of good chips out of a wafer).

Operational faults (or just "faults") have in comparison a considerably lower probability of occurrence, the difference of which may be in orders of magnitude. Improvements in the solid-state technology and maturity of the fabrication processes have reduced the failure rate of a single component within a VLSI chip. However, the exponential increase in the component-count per VLSI chip has more than offset the increase in reliability of a single component. Thus, operational faults cannot be ignored although they have a substantially lower probability of occurrence compared to production defects. Consequently, a fault-tolerance strategy that enables the system to continue processing, even in the presence of operational faults, can be beneficial.

The two types of failures, manufacturing defects and operational faults, also differ in the costs associated with them. Defects are tested for before the ICs are assembled into a system and therefore, they contribute only to the production costs of the ICs. In contrast, faults occur after the system has been assembled and is already operational. Hence, their impact is on the system's operation and their damage might be substantial, especially in systems used for critical real time applications. Clearly, a method which is cost-effective for handling defects is not necessarily cost-effective for handling operational faults, and vice versa.

For both types of failures in VLSI, a repair operation is impossible and the best one can do is to somehow avoid the use of the faulty part by restructuring the system. This implies that in the wafer (in the case of defects) or in the assembled system (in the case of faults) there are other operational parts which are either identical to the faulty one or that can fulfill the same tasks.

Restructuring can be static or dynamic. Static restructuring schemes are suitable only to avoid the use of parts with production flaws. Dynamic restructuring is required during the normal system operation, when faulty parts have to be restructured out of the system without human intervention. Such a dynamic strategy might be appropriate to handle defects as well. Static schemes tend to use comparatively less hardware but consume operator time, while dynamic schemes are controlled internally by the system and usually require extra circuitry.

Another aspect that has to be considered when evaluating the effectiveness of a given fault-tolerance technique, is the required hardware investment. The hardware added can be in the form of switching elements, (e.g., [8], [38] and [41]) or redundancy in processors or communication links (e.g., [10], [26]). When carrying out such an analysis we have to take into account the following two parameters:

(1) The relative hardware complexity of processors, communication links and switching elements (if they exist).

(2) The susceptibility to failures (manufacturing defects or operational faults) of all the above-mentioned elements.

Processing elements are traditionally considered the most important system resource; hence, achieving 100% utilization of them is many times attempted. For example, in [8], [38] and [41] switching elements are added between processors to assist in achieving this goal. In [10] and [26] connecting tracks are added on the wafer to be used in bypassing the defective PEs when connecting the fault-free ones. However, the silicon area that needs to be devoted to switching elements (e.g., switches capable of interconnecting 4 to 8 separate parallel busses [41]) or to additional communication links cannot be ignored. Consequently, such schemes might be beneficial only for PEs which are substantially larger than the switches and the additional links (e.g., [32]). Also, the addition of switching elements and especially the longer interconnections between active processors result in longer delays affecting the throughput of the system. To overcome this performance penalty, it has been suggested in [25] to add registers for bypassing faulty processors. The effect of this is to introduce extra stages in the pipeline thus, increasing the latency of the pipeline without reducing its throughput.

In the above mentioned schemes, one of the underlying assumptions is that the extra circuitry (e.g., switching elements, communication links or registers) are failure-free and only processors can fail. However, larger silicon areas devoted to those elements increase their susceptibility to defects or faults; as a result, the above-mentioned assumption might not be valid any more.

In VLSI, the silicon area devoted to a system element might be more important than its hardware complexity. Consequently, 100% utilization of PEs is not necessarily the major objective, especially if this requires adding switches and/or communication links, which consume silicon real-estate. In the new technology, processors will be the expendable components, as gates were in SSI or small logic networks in LSI.

This may justify different fault-tolerance schemes which do not attempt to achieve 100% utilization of the fault-free processors when the array is restructured to avoid the use of faulty ones [18]. Such schemes, which give up the use of some fault-free PEs upon restructuring, can be attractive for operational faults (which are few in number). Here, the lack of additional hardware (switches or links) allows a larger number of PEs to fit into the same chip area, thereby offsetting the penalty of giving up the use of fault-free PEs when restructuring.

The reported research in this area of fault-tolerant architectures, although a significant beginning, is limited in the following aspects:

(a) Most of the proposed architectures have been developed on an ad-hoc basis. No well-established criterion or framework yet exists for the formulation of these architectures.

(b) As indicated above, redundancy can be used for both yield enhancement and reliability improvement. Recently, development of models to evaluate how can a given redundancy be shared to achieve the best combined improvement of yield and performance has begun [21] but more extensive work is still needed. Such models could be also used to compare and evaluate different architectures.

(c) The testability and reconfigurability issues have seen very limited treatment. Algorithms for testing, diagnosis and reconfiguration need to be developed.

III. A TAXONOMY FOR MULTI-PROCESSOR ARCHITECTURES

Broadly, there are two types of interconnection architectures that are of interest to VLSI processor array implementation. The first type is the nearest neighbor interconnection which includes various mesh interconnections, illustrated in Fig. 1. The second type we refer to here as algebraic graph networks which includes networks such as binary n-cube, cube-connected cycles, shuffle-exchange graph , shift-and-replace graph networks and group graph networks. Examples of the latter are illustrated in Fig. 2. Like the mesh connection networks, these admit efficient execution of certain algorithms. Also algebraic structure of some of these networks can be exploited so as to realize asymptotically optimum VLSI-layouts.

In order to represent uniformly different types of such architectures, using different types of processing nodes (processors with internal switches and processors with external switches) and different types of switches (switches used for routing and switches used for fault-detection and reconfiguration), we present the following taxonomy. Generally, there are two types of system nodes: nodes capable of only computation, and nodes capable of both computing and switching for routing. In addition, there are two types of switches, the conventional switches, capable of only establishing connections, and fault-detecting switches, those that perform the function of both fault-detection and reconfiguration. Different types of architectures are delineated in Fig. 3. The advantage, generally, in using external switches is that the computational space can be distinct from the communication space which therefore, provides greater flexibility for emulation of a variety of communication geometries. The disadvantage of external switches, though, is that they require additional hardware support and occupy extra VLSI area.

Different types of architectures are illustrated in Fig. 3. First, Fig. 3(a) illustrates an architecture where the PEs perform internally all the switching necessary to establish connections. Fig. 3(b) represents an architecture where all the connections are established by using external switches. Such differences are best illustrated by using the following 5-tuple representation of networks. Let $N = \langle P, S, E_p, E_s, E_{p-s} \rangle$ denote the network, where: P represents the set of PEs, S denotes the set of switches, E_p denotes the set of direct processor-processor links, E_s denotes the set of direct switch-switch links and E_{p-s} denotes the set of processor-switch links. Different architectures can be conveniently categorized into the following four types, as shown below, where ϕ represents the null set:

Type 1: $\langle P, S = \phi, E_p, E_s = E_{p-s} = \phi \rangle$

This denotes the type of architecture shown in Fig. 3(a). Here, the array contains only processing no
part of the processor. The mesh connections considered in [18] is an example of such an architecture.

Type 2: $\langle P, S, E_p = \phi, E_s, E_{p-s} \rangle$

This denotes the type of architecture shown in Fig. 3(b) where all of the configuration and communication functions are performed by switches that are external to the processor. The CHIP architecture proposed by Snyder [41] is an example of this type.

Type 3: $\langle P, S, E_p, E_s = \emptyset, E_{p-s} \rangle$

Fig. 3(c) delineates such an architecture. Here, in addition to the external switches, each processor has an internal switch which sets up the connections between processors. The external switches are used to provide the function of fault-detection through disagreement detection and subsequent switching out of the faulty processor, thus disconnecting it from the network.

Type 4: $\langle P, S, E_p, E_s, E_{p-s} \rangle$

This denotes a type of architecture where all of the different types of links are used. An example of such an architecture is illustrated in Fig. 3(d). Here, a linear array of PEs is provided with external switch connections which can be configured in four ways, as shown in Fig. 4(a). The switches in such an architecture have a dual purpose. First, they can be used to provide multiple logical configurations such as binary tree in addition to the linear array; thus, an application that requires both linear array and binary tree can use this architecture as shown in Fig. 4(b). Secondly, the switches can be used to bypass the faulty elements as shown in Fig. 4(c).

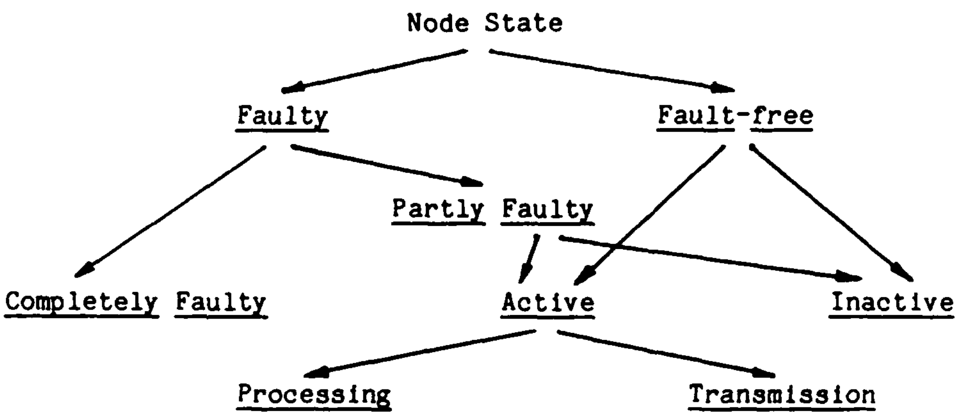
Thus as we see, these different categorizations encompass all of the different possible architectures that can be conceived. Therefore, the above taxonomy provides a convenient framework for both the analysis of different architectures as well a for the conceptualization of new architectures.

There are two basic ways one can introduce fault-tolerance into these arrays, the first approach would be to provide redundancy at each node so that the node can be reconfigured internally in the event of a fault. For example, consider a 9-node mesh connection shown in Fig. 5. If we assume that the interconnects are highly reliable, one way to design this array so that it will be fault-tolerant is to use two self-checking processors at each node, as shown in Fig. 6. The function of the external switch is to determine, in the event of a fault, which one of the two checkers is indicating errors and then switch out the appropriate module.

However, if the interconnects cannot be assumed to be reliable, one has then to provide redundancy by designing an array larger than the maximum size required for the applications. For example, consider the 4x4 array shown in Fig. 7 which is designed to support various applications including the binary tree configuration shown in Fig 8(a). The mapping of the binary tree onto the array, is depicted in Fig. 8(b). In this figure, the mapped nodes of the binary tree are shown, along with the inactive components, which are shown using dashed lines. Consider now that the active node 6 becomes faulty. It can be easily seen that the network can no longer admit the binary tree configuration, shown in Fig. 8(a). However, should it be possible to execute the same application on a reduced binary tree (perhaps with a degraded performance) such as the one shown in Fig. 9, the application can still be supported by the faulty array, as demonstrated below.

There are two different ways this can be achieved. First, the original 4x4 array can be restructured into a smaller 3x3 array as shown in Fig. 10. This would require giving up the use of some processing nodes by turning them into connecting elements (CEs) [18]. Then, any application that can be executed on a 3x3 array can be executed on this new (logical) 3x3 array. The second approach would be to map directly the application configuration onto the faulty physical array. However, the latter approach can be computationally complex [9]. Thus, depending on whether or not such reduction is possible, the network may or may not be fault-tolerant, with respect to this application.

Several important concepts emerge from the above discussion. Firstly, a node or link can assume several distinct states. The following shows various possible states of the node:



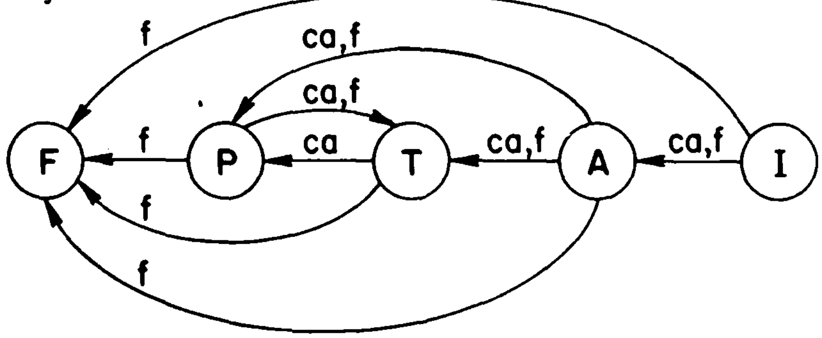
Here, the processing state of the node refers to that state in which the node is assigned to perform some useful computational task.

On the other hand, a node in the transmission state is assigned to perform only switching, so as to establish a path. Thus, a node in this state does not perform any computations, except those which may be required for routing, etc. For a link though, this distinction does not apply. Accordingly, there are fewer states for a link, as shown below:

Switch/Link State

- Faulty
- Fault-free
  - Active
  - Inactive

The various possible state transitions are shown by the following directed graph. Here, F,P,T,A and I denote the faulty, processing, transmission, active and inactive states, respectively. The arc labels, f and ca, represent the transitions caused by fault, and change of application, respectively.



Secondly, the various reconfiguration processes can be conceptualized through an abstraction of layers, formulated below:

Let the <u>physical layer</u> represent the topology which describes the interconnection structure, along with the status of the nodes and links in the physical array. A node/link in the physical layer can be either in the fault-free or faulty state.

Let an application layer represent that topology which is required to support a given application. Thus, in this layer, all of the nodes are processing nodes; the links, active links.

Let the logical layer represent the topology which realizes a given application layer on a given physical layer. Thus, a node in this layer is either in the processing state or in the transmission state. All of the links in the logical layer are in the active state.

For a given configuration, the above layers are related topologically, as shown in Fig. 11. The nodes in the application layer are a subset of the nodes in the corresponding logical layer and these are in turn a subset of the nodes in the physical layer.

The following defines a set of fundamental problems of practical importance:

Problem 1: Given an application layer (a set of application layers) and the physical array that admits these application(s), what is the minimum size (number of nodes, silicon area) of the physical layer that can admit the applications(s) when t or fewer component fail?

Problem 2: Given the geometrical structure(s) of an application layer (set of application layers), how can a physical array be designed so that it can provide "efficient" fault-tolerant realization of the application(s)? The term efficient may be defined in terms of factors such as size of physical array, length of communication delay between adjacent application nodes, ease of testing and diagnosis, reconfigurability, etc.

The above problems need to be studied in the context of more general and flexible use of redundancy. For example, judicious use of node-level redundancy may offset the need for massive redundancy at the system level. Also, broader use of switches as implied by Type 3 and Type 4 architectures may yield new system architectures - architectures that provide more efficient utilization of redundancy.

These above discussions are also applicable to the second type of networks, the algebraic networks. For example, consider the shift-and-replace graph networks proposed recently in [39] as a candidate for VLSI processor networks. Such an 8-node network is shown in Fig. 12(a). This network is capable of emulating various useful logical structures such as the linear array, binary tree, shuffle and the shuffle-exchange communication structures, as shown in Fig. 12(b). More importantly, this algebraic network can emulate structures such as the linear array and binary tree, in spite of a fault. For example, consider the link connecting nodes 1 and 2 becoming faulty. In this case, the networks can still be restructured both as a linear array and as a binary tree, as shown in Fig. 13. Similarly, the network is also capable of emulating these structures in spite of any single node failures.

It may also be noted that networks such as the binary n-cube and the cube-connected cycles provide some interesting fault-tolerant reconfiguration capabilities. For example, consider a 4-cube of 16 nodes, shown in Fig. 14(a). In the event of a fault, one can degrade this to a 3-cube of 8 nodes, as illustrated in Fig. 14(a). However, this would require giving up the use of seven good nodes. Alternatively, one can partition the 4-cube into 4 subnetworks of 2-cubes. Assuming that the problem can be divided into subproblems that can be executed on 2-cubes, one can use 3 of these, as shown in Fig. 14(b). This would necessitate giving up the use of only 3 good nodes. It is obvious that the fault-tolerance of algebraic networks can be studied in the context of VLSI processor arrays.

IV. TESTING AND RECONFIGURATION STRATEGIES

Central to the success of any fault-tolerance scheme is the formulation of effective testing and reconfiguration strategies. Basically there are two different approaches to diagnosis and recovery: centralized and distributed. In a centralized procedure, one may assume an external unit which is responsible for initiating testing and reconfiguration. In a distributed procedure, the PEs, themselves, are responsible for performing periodic testing and reconfiguration.

The advantage of a centralized scheme is that no additional hardware and software support have to be provided within each PE to allow testing and reconfiguration. On the other hand, useful computation for the entire array has to be interrupted so that testing can be performed. Additionally, the complexity of the circuit and the limited access from the external unit may not allow a centralized procedure to be used. The advantage of distributed testing, on the other hand, is that since each processor can perform testing in an asynchronous mode, the testing can be interleaved with computation, thus not necessarily requiring a complete interruption of all useful computation. Moreover, the distributed testing has the potential for better fault coverage because of the proximity of the testing unit and the unit under test.

From the above discussion, it is apparent that a distributed procedure must strive to make the testing and reconfiguration task local to each node. This way, the testing and reconfiguration can be made transparent to most of the network. However, performing these tasks locally requires extra hardware and software support at each node and a distributed procedure must try to minimize it. On the other hand, a centralized procedure must attempt to minimize the number of tests that will be required when no faults are present. Interruption of useful computation will be this way minimized.

In the following, we present an example for a distributed testing procedure in which every PE tests all its immediate neighbors. In this way, faulty PEs and faulty connections between PEs are detected by the adjacent PEs. The procedure first partitions all the PEs into m disjoint testing groups, T_0, T_1,..., T_{m-1}. After this partitioning, there are m phases of testing, where at phase i ($0 < i < m-1$), the members of T_i test all their neighbors.

The partition is such that (1) every PE is surrounded by PEs of other groups, and (2) no PE has two neighbors belonging to the same group. These two properties guarantee that for every i, no two members of T_i will test each other, or try simultaneously to test a third PE. It can easily be shown that five (seven) groups are both necessary and sufficient for a partition with the above properties in the case of a square array [18] (hexagonal array [12]). The testing group numbers assigned to each PE in a square array and an hexagonal array may be calculated from its array indices (p,q) by (p+2q) mod 5 and (p+2q) mod 7 , respectively.

When all the m phases of the testing procedure have been completed, each and every PE knows the status (faulty/not-faulty) of all its immediate neighbors and the corresponding connecting links. There is no difference if the actual fault is in the neighboring PE proper, or in the link leading to it.

Moreover, the status of a faulty PE or link will be known only to its neighboring PEs. This locally stored information is sufficient for a distributed reconfiguration algorithm (e.g., [18]) that will follow the testing procedure. Thus, it may be seen that the above distributed testing procedure does not require any passing of test results, as required in other, more general, distributed diagnosis algorithms (e.g., [22]), by taking advantage of the regularity of the VLSI array.

It may be noted that the above algorithm will also work with simple comparison testing. In this type of testing, there are no tests to be applied from one processor to the other. Simply, what is required is that two neighboring processors, i and j, exchange the results of certain predetermined identical computation. In the event that there is a mismatch, processor i can assume j is faulty and processor j can assume i is faulty.

In summary, a key feature of the above distributed testing procedure is that the testing and subsequent reconfiguration are transparent to all the nodes in the network except for those that are adjacent to the faulty node. The main disadvantage of distributed procedures is however, the extra hardware and software support that each PE must provide for testing and reconfiguration. This may be difficult to accomplish in processing arrays consisting of very small and simple PEs.

As discussed earlier, centralized testing may have to interrupt all the computations in the array. Since it is assumed that the testing is done periodically, it is desirable that the number of tests and the testing time should be minimized when there are no faults. The testing time should be proportionate to the number of faults; thus a fault-free array would require a minimum number of tests with the number of tests increasing with the number of faults. In [31], a possible diagnosis strategy was suggested that makes the testing very simple in the absence of any fault; the testing becomes progressively more time-consuming with the number of faults. Since most of the time no faults are present, the performance penalty due to interruption for testing can be minimal. This is illustrated further below.

In Fig. 15 possible testing graphs for a 5*5 end-around mesh (the boundary nodes are also adjacent) are shown. The darkened boxes represent nodes already diagnosed as being faulty. The edges with arrows indicate those communication edges included in the testing graph. The arrows point from the tester to the tested unit. Algorithm SELF2 [22] would require a graph with 75 directed edges to diagnose up to three faults. The strategy presented in [31] never employs more than 25 periodic tests.

Fig. 15(a) indicates a possible initial testing graph. Since the end-around mesh is node-symmetric, the first fault may always be viewed as occurring in the center node; and the same testing graph may then be used after the first fault is diagnosed. There must exist two adjacent fault-free rows (also columns) after no more than two faults have occurred. This ensures the graph may be viewed with the faults restricted to the interior, i.e., with the border intact.

Figures 15(b) through 15(f) illustrate five possible cases for the fault locations. In each instance the interior is shown to include a Hamiltonian path. As proved in [31], at least one fault amongst the nodes in the loop along the border may be diagnosed. If all are fault-free, then the first faulty node along the path through the interior may be diagnosed.

Let $[\alpha, \beta]$ denote the closed interval from α to β. Let the nodes in the mesh be represented by pairs <a,b> where a, b $\in [1,5]$ with a indicating the row and b indicating the column. Let the first fault, without loss of generality, be at node <3,3>. By symmetry we need only to consider the second fault occurring at (1) <2,4>, (2) <2,3>, (3) <1,5>, (4) <1,4>, or (5) <1,3>. These possibilities (1) through (5) correspond to the illustrations in Figures 15(b) through 15(f), respectively. Consequently, Fig. 15 gives testing graphs for all unique fault patterns in this case. Precise necessary and sufficient conditions for such a dynamic testing of general systems are given in [31].

V. ANALYTICAL MODELS FOR EVALUATION OF YIELD AND PERFORMANCE

The introduction of fault-tolerance into the architecture of VLSI-based multiprocessor systems has two objectives. One is yield enhancement, the other is improvement of performance. To achieve these two goals, redundancy has to be introduced either at the basic element level or/and at the system level. In the latter case, redundant elements can be added to the original design and they will be used to replace defective ones after the manufacturing process has been completed. Such a replacement is done by reconfiguring the system using either a static scheme or a dynamic one. Once this procedure is completed the system goes into operation and it has to handle from this point on only operational faults. This can be done using a dynamic reconfiguration scheme which might be different from the one used for defects. At this point the fault-tolerance capacity of the system is used to improve its performance. First, the remaining redundant elements (if any) can be used as spares and then, the system is gracefully degraded. We conclude therefore, that the same redundancy can be used for yield enhancement and for performance improvement as well.

We present in this section an analytical model that enables us to consider both manufacturing defects and operational faults. This model allows us to analyze the effectiveness of a given fault-tolerance technique in increasing yield and improving performance, or find the tradeoff between the two. It also enables us to compare various fault-tolerance techniques, examine different system topologies and determine the optimal amount of redundancy to be added.

To formulate such a model an expression for the yield of a fault-tolerant multiprocessor chip is needed. Such expressions have been presented in [20] and [28]. A more general expression for the yield was proposed in [21] and is presented in what follows.

The yield of any VLSI chip depends on the types of defects which may occur during the manufacturing process and their distribution. The majority of fabrication defects can be classified as random spot defects [43] caused by minute particles deposited on the wafer. Hence, each of them may affect only a single element (like a processor, bus, etc.) in a multiprocessor chip.

For the statistics of the fabrication defects we can adopt one of the models suggested in the literature like Poisson, general negative binomial, binomial statistics and others. Under proper assumptions each one of these statistics can be used and the "correct" one is the one that fits the data best [43]. One model which has been shown to agree with experimental results, is the generalized negative binomial distribution [42]. Its attractiveness stems from the fact that it does not assume that all defects are evenly distributed throughout the wafer but rather allows defects to cluster. The probability of having x defects on a chip for this distribution is,

$$Pr\{X = x\} = \frac{\Gamma(x+\alpha)}{x!\,\Gamma(\alpha)} \cdot \frac{\left[\frac{\bar{\lambda}}{\alpha}\right]^{x}}{\left[1 + \frac{\bar{\lambda}}{\alpha}\right]^{\alpha+x}} \tag{1}$$

where $\bar{\lambda}$ is the average number of defects per chip and α is the defect clustering parameter. A low value of α can be used to model severe clustering of defects on a wafer, while for $\alpha \to \infty$ we obtain the Poisson distribution. This two-parameter distribution has a mean of $\bar{\lambda}$ and a variance of $\bar{\lambda}(1 + \bar{\lambda}/\alpha)$. The mean and variance of data obtained from many wafer samples are used to estimate these two parameters.

For non-redundant chips the yield is the probability of having zero defects,

$$Y = Pr\{X=0\} = \left[1 + (\frac{\bar{\lambda}}{\alpha})\right]^{-\alpha} \tag{2}$$

Suppose now that redundancy is added to a chip so that s defective elements can be tolerated (i.e., replaced by good spares), and denote by N the total number of elements (e.g., processors). Then, the chip is acceptable with any number of manufacturing defects as long as all of them are restricted to at most s elements. The yield, which is now the probability of a chip being acceptable, is given by,

$$Y = \sum_{x=0}^{\infty} \Pr\{\text{ There are x defects in at most s elements }\} \tag{3}$$

If we denote,

$$Q_{x,i}^{(N)} = \Pr\{\text{ x defects are distributed into exactly i out of N elements / There are x defects }\}$$

Then,

$$Y = \sum_{x=0}^{\infty} \sum_{i=0}^{s} Q_{x,i}^{(N)} * \Pr\{\text{There are x manufacturing defects in the chip}\} \tag{4}$$

The last term in the above equation is $\Pr\{X=x\}$ and we may substitute it by equation (1) or a similar expression for any other defect distribution (e.g., Bose-Einstein statistics [28]).

The probability $Q_{x,i}^{(N)}$ is given by,

$$Q_{x,i}^{(N)} = \sum_{k=0}^{i} (-1)^k \binom{N}{k, i-k, N-i} \left[\frac{i-k}{N}\right]^x \tag{5}$$

where $\binom{N}{k, i-k, N-i}$ is the multinomial coefficient.

In the previous discussion we have assumed that only one type of elements can have defects. If two types of elements (e.g., processors and communication busses) can have defects, then the probability of having x_1 defects in type 1 elements and x_2 defects in type 2 elements is,

$$\Pr\{X_1=x_1,\ X_2=x_2\} = \Pr\{X_1=x_1\} * \Pr\{X_2=x_2\} \tag{6}$$

since the probability of defects in different types of elements are independent [43].

Suppose now that s_1 defective elements of type 1 and s_2 defective elements of type 2, out of N_1 and N_2 elements, respectively, can be tolerated. Then, the yield is given by,

$$Y = \sum_{x_1=o}^{oo} \sum_{x_2=o}^{oo} \left| \sum_{i_1=o}^{s_1} Q_{x_1,i_1}^{(N_1)} \right| \left| \sum_{i_2=o}^{s_2} Q_{x_2,i_2}^{(N_2)} \right| * Pr\{X_1=x_1, X_2=x_2\} \tag{7}$$

s_1 and s_2 are not necessarily independent; for example, if less than s_1 elements of type 1 are defective we may be able to tolerate more than s_2 defective elements of type 2. Equation (7) will have in this case to be changed accordingly.

Equation (7) as well as equation (4) can be multiplied by a "bypass coverage probability" [28]. This is the conditional probability that an element can be bypassed given that it is faulty. By adding this probability one may consider less than perfect procedures for locating faulty elements and reconfiguring them out of the system.

In the following we adopt the commonly used assumption that only one type of elements can fail (usually, the more complex one, e.g., the processors). The general case in which all system elements can have defects in them, can be analyzed based on expressions similar to (7).

To tolerate s defective elements, at least s redundant ones are needed. However, the exact amount of required redundancy depends upon the specific static or dynamic reconfiguration scheme used. This in turn, determines the increase in chip area which must be taken into account when calculating the yield, since a larger number of defects is expected now.

Let γ_s denote the increase in chip area (due to the addition of redundancy) needed to tolerate these s faulty elements. The factor γ_s is called the redundancy factor [20] and it depends on the system topology and the reconfiguration strategy. To take into account the increased number of expected defects, we have to substitute $\bar{\lambda}$ (the average number of defects per chip) by $\gamma_s\bar{\lambda}$ in equation (1).

In addition, any increase in chip area will reduce the number of chips that will fit into the same wafer. Hence, instead of calculating the yield which is the probability that a single chip is acceptable, one has to calculate the expected number of acceptable chips out of a given wafer. This expression, called equivalent yield in [20], is obtained from equation (4) after dividing it by γ_s. By comparing the equivalent yield of the fault-tolerant chip and the yield of the simplex one, we can determine whether it is beneficial when yield is considered, to have built-in fault-tolerance and how many redundant elements should we add. This comparison can be done for various topologies of multi-processors and different reconfiguration algorithms.

An analysis along these lines has been done in [28] and in [20]. In both it has been observed that the improvement in yield saturates above some amount of redundancy. This indicates that there is an optimal amount of redundancy that should be added.

Chips having s or less defects will be accepted and then reconfigured to avoid the use of the defective elements. If the number of defects was less than s , the chip has some "residual" redundancy which can then be used for performance enhancement, i.e., handle operational faults which occur during the life time of the system. Even chips in which no redundant elements are left when leaving the manufacturing site (i.e., there were originally s defects in the chip), can still benefit from the fault-tolerance capability.

To evaluate the effectiveness of the "residual" redundancy and the fault-tolerance capacity of the chip we have to select some performance measures and we need a model that will allow us to calculate these measures. A natural choice for this purpose is a Markov model like the one employed by [20] and [6].

Suppose first that the same reconfiguration scheme is used to avoid manufacturing defects and operational faults as well. This assumption implies that a dynamic scheme is employed since no static scheme can be used while the system is in operation. The suggested Markov model for this case is depicted in Figure 16, where (F) is the system failure state and (j) is a state at which the system is operational in the presence of j faulty elements. A transition from state (j) to state (F) takes place when an additional node becomes faulty and the system fails to recover from its effect. The corresponding transition rate is denoted by α_j^F. Similarly, α_j^{j+1} is the transition rate from state (j) to state (j+1). These transition rates depend upon the failure rates of the system's elements and the coverage probability [20].

State (o) in Figure 16 is the initial state of the system if no defects occurred while the chip has been manufactured. If there were i defective elements ($o \leq i \leq s$) then (i) will be the initial state. Let a_i denote the probability of this event [21],

$$a_i = \sum_{x=0}^{\infty} Q_{x,i}^{(N)} * Pr\{X=x\} \tag{8}$$

Using a_i we can calculate the yield as

$$Y = \sum_{i=0}^{s} a_i \tag{9}$$

State (s+m) in Figure 16 is a terminal state [20] (i.e., a state from which the only transition possible is to the system failure state (F)), where m is the largest number of faulty elements that the system can tolerate if no redundant elements were left when the system went into operation.

Let $P_j^i(t)$ = Pr{ The system is in state (j) at time t / The system was initially in state (i) }

$$i = 0,1,\ldots,s; \quad j = i,i+1,\ldots,s+m$$

with $P_i^i(0)=1$ and $P_j^i(0)=0$ for $j>i$.

The Markov model in Figure 16 is described then by the following differential equations:

$$\frac{dP_i^i(t)}{dt} = -\alpha_i P_i^i(t) \tag{10}$$

$$\frac{dP_j^i(t)}{dt} = -\alpha_j P_j^i(t) + \alpha_{j-1}^j P_{j-1}^i(t) \tag{11}$$

where $j= i+1,i+2,\ldots,s+m$ and

$$\alpha_j = \alpha_j^F + \alpha_j^{j+1}$$

The solution of equations (10) and (11) under the condition

$$\alpha_j \neq \alpha_k \quad \text{for all} \quad (k) \neq (j)$$

which is satisfied in most practical cases, is

$$P_j^i(t) = \alpha_i^{i+1} \alpha_{i+1}^{i+2} \dots \alpha_{j-1}^{j} \sum_{u=i}^{j} \frac{e^{-\alpha_u t}}{\prod_{\substack{v=i \\ v \neq u}}^{j} (\alpha_v - \alpha_u)} \tag{12}$$

and

$$P_i^i(t) = e^{-\alpha_i t} \tag{13}$$

For the Markov model shown in Figure 16 we can calculate several performance measures like Reliability, Performability, Computational availability and Area utilization [20]. Let $R_i(t)$ $(0 \leq i \leq s)$ denote the reliability of a system (i.e., the probability that it operates correctly in the time interval [o,t]) which had i defects during the manufacturing process. This reliability can be calculated from the above Markov model as follows,

$$R_i(t) = \sum_{j=i}^{s+m} P_j^i(t) \tag{14}$$

We may then define and compute

$$R(t) = \frac{1}{Y} \sum_{i=o}^{s} a_i \, R_i(t) \tag{15}$$

as the average reliability of a system having s or less defects when manufactured. This average reliability can then be compared to $R_s(t)$ which is the reliability of a system with no redundancy left from the manufacturing step. If we set s=0 then $R_o(t)$ is the reliability of the system if only perfect chips (with no defects) are accepted.

Similarly, we can define and calculate the computational availability $A_c^i(t)$ (the expected available computational capacity) and area utilization measure $U_i(t)$. The latter takes into account the additional area needed when fault-tolerance is introduced into the system, and is defined in the following way,

$$U_i(t) = \frac{\text{Computational Availability } A_c^i(t)}{\text{Chip area increase } \gamma_s}$$

The expression for the above introduced computational availability measure is,

$$A_c^i(t) = \sum_{j=i}^{s+m} c_j P_j^i(t) \tag{16}$$

where c_j is the computational capacity of the system in state (j) [20], expressed for example in instructions per time unit. The computational capacity depends mainly on the number of processors available for computation in state (j). This number is at most N-j processors (where N is the number of processors in the fault-free system), and is determined by the reconfiguration strategy. In addition, c_j depends on the current system structure and application since not all processors are utilized in every possible structure or application.

Other performance measures, like mean time to failure, can also be calculated. For example, let T_i denote the mean time to failure of a system which was initially in state i, then

$$T_i = \int_0^\infty R_i(t)\, dt \tag{17}$$

The average mean time to failure can be defined similarly to equation (15).

This model can be extended in two directions in order to make it more general and more practical. One is to include two or more types of system elements that can fail (during manufacturing or later on) like communication busses, switches etc. The second one is to allow the use of one reconfiguration scheme to handle defects and a different one to handle operational faults. Manufacturing defects can be effectively handled using static schemes like "laser programming" or electrically fusible links, while operational faults are best handled by some dynamic reconfiguration scheme. A static scheme for defects requires less silicon area on one hand but consumes operator time on the other hand. A more general Markov model with two different reconfiguration schemes will enable us to analyze the effectiveness of various such schemes.

Using the method presented in [20] one can derive closed-form expressions for the state probabilities and compute the yield and various performance measures for different architectures.

VI. CONCLUSIONS

Fault-tolerant architectures that use redundancy for yield and performance improvement have been considered. We have presented a unified framework through which existing architectures incorporating fault-tolerance can be analyzed and new ones suggested.

Several problems related to testing and reconfiguration of these arrays have been described. Both the distributed and centralized modes of testing have been considered.

The last part of the paper is devoted to the presentation of analytical models for the evaluation of reliability and yield improvement through redundancy. The available redundancy on the chip or wafer is primarily limited by the size of the chip or wafer hence, it is imperative to find a method by which one can optimally share the available redundancy between yield enhancement and performance improvement. The models discussed can be used to study the effect of sharing available redundancy between these two somewhat competing requirements.

VII. REFERENCES AND BIBLIOGRAPHY

[1] D.P. Agrawal, "Testing and Fault-tolerance of Multistage Interconnection Network," Computer, Vol.15, April 1982, pp.41-53.

[2] R.C. Aubusson and I. Catt, "Wafer-Scale Integration - a Fault-Tolerant Procedure," IEEE J. Solid-State Circuits, Vol. SC-13, June 1978, pp.339-344.

[3] M.D. Beaudry, "Performance-Related Reliability Measures for Computing Systems", IEEE Trans. on Computers, Vol. C-27, June 1978, pp. 540-547.

[4] W.C. Carter, et al., "Cost Effectiveness of Self-Checking Computer Design", Proc. of the 7th Symposium on Fault-Tolerant Computing, June 1977, pp. 117-123.

[5] R.P. Cenker et al., "A Fault-Tolerant 64K Dynamic Random-Access Memory", IEEE Trans. on Electron Devices, Vol. ED-26, June 1979, pp. 853-860.

[6] J.A. Fortes and C.S. Raghavendra, "Dynamically Reconfigurable Fault-Tolerant Array Processor," Proc. of the 14th Annual Symposium on Fault-Tolerant Computing, June 1984, pp. 386-392.

[7] M.J. Foster and H.T. Kung, "The design of special-purpose VLSI chips," Computer, Vol. 13, January 1980, pp. 26-40.

[8] D. S. Fussel and P. J. Varman, "Fault-Tolerant Wafer-Scale Architectures for VLSI," Proc. of the 9-th Annual Symp. on Comp. Arch., May 1982.

[9] P.J. Varman and D.S. Fussel, "Realizing Fault-Tolerant Binary Trees in VLSI," typescript, University of Texas at Austin, 1983.

[10] J.W. Greene and A. El Gamal, "Configuration of VLSI Arrays in the Presence of Defects," Journal of the ACM, Vol. 31, No. 4, Oct. 1984, pp. 694-717.

[11] D. Gordon, I. Koren, and G. M. Silberman, "Embedding Tree Structures in VLSI Hexagonal Arrays," IEEE Trans. on Computers, Vol. C-33, Jan. 1984, pp.104-107.

[12] D. Gordon, I. Koren and G.M. Silberman, "Fault-Tolerance in VLSI Hexagonal Arrays," typescript, Dept. of Electrical Engineering, Technion, Haifa, Israel.

[13] J.P. Hayes, "A Graph Model for Fault-tolerant Computing System," IEEE Trans. on Computers, Vol. 25, September 1974, pp. 875-884.

[14] L.S. Haynes, R.L. Lau, D.P. Siewiorek and D.W. Mizell, "A Survey of Highly Parallel Computing," Computer, Vol. 15, January 1982, pp. 9-24.

[15] K. Hedlund and L. Snyder, "Wafer Scale Integration of Configurable, Highly Parallel (Chip) Processor," Proc. of the 1982 Internl. Conf. on Parallel Processing, August 1982, pp. 262-265.

[16] E. Horowitz and A. Zorat, "The binary tree as an interconnection network: Applications to Multiprocessor Systems and VLSI," IEEE Trans.

on Computers, Vol. C-30, April 1981, pp. 247-253.

[17] K.H.J. Huang and J.A. Abraham, "Low Cost Schemes for Fault-tolerance in Matrix Operations with Array Processors," Proc. of the 12th Symp. on Fault-Tolerant Computing, June 1982.

[18] I. Koren, "A Reconfigurable and Fault-tolerant VLSI Multiprocessor Array," Proc. of the 8th Annual Symposium on Computer Architecture, May 1981, pp. 425-441.

[19] I. Koren and G.M. Silberman, "A Direct Mapping of Algorithms onto VLSI Processor Arrays Based on the Data Flow Approach," Proc. of the 1983 Internl. Conf. on Parallel Processing, August 1983, pp. 335-337.

[20] I. Koren and M.A. Breuer, "On Area and Yield Considerations for Fault- Tolerant VLSI Processor Arrays," IEEE Trans. on Computers, Vol. C-33, Jan. 1984, pp. 21-27.

[21] I. Koren and D.K. Pradhan, "Introducing Redundancy into VLSI Designs for Yield and Performance Enhancement," Proc. of the 15th Annual Symposium on Fault-Tolerant Computing, June 1985, pp. 330-335.

[22] J.G. Kuhl and S.M. Reddy, "Distributed Fault-Tolerance for Large Multiprocessor systems," Proc. of the 7th Symposium on Computer Architecture, May 1980, pp. 23-30.

[23] H.T. Kung, "The Structure of Parallel Algorithms," Advances in Computers, Vol. 19, M.C. Yovits ed., Academic Press, NY, 1980, pp. 65-112.

[24] H.T. Kung, "Why Systolic Arrays," Computer, Vol. 15, January 1982, pp.37-46.

[25] H.T. Kung and M.S. Lam, "Fault-Tolerance and Two-Level Pipelining in VLSI Systolic Arrays," Proc. of 1984 Conf. on Advanced Research in VLSI, MIT, Jan. 1984, pp. 74-83.

[26] F.T. Leighton and C.E. Leiserson, "Wafer-Scale Integration of Systolic Arrays," IEEE Trans. on Computers, Vol. C-34, May 1985, pp. 448-461.

[27] A.D. Malony, "Regular Interconnection Networks," Tech. report CSD-82-0825, Dept. of Computer Science, UCLA, 1982.

[28] T.E. Mangir and A. Avizienis, "Fault-Tolerant Design for VLSI: Effects of Interconnect Requirements on Yield Improvements of VLSI Designs," IEEE Trans. on Computers, Vol. C-31, July 1982, pp. 609-615.

[29] F.B. Manning, "An Approach to Highly Integrated Computer-Maintained Cellular Arrays," IEEE Trans. on Computers, Vol. C-26, June 1977, pp.536-551.

[30] C. Mead and L. Conway, "Introduction to VLSI Systems," Addison-Wesley, 1980.

[31] F.J. Meyer and D.K. Pradhan, "Dynamic Testing Strategy for Distributed Systems," Proc. of the 15th Annual Symposium on Fault-Tolerant Computing, June 1985, pp. 84-90.

[32] H. Mizrahi and I. Koren, "Evaluating the Cost-Effectiveness of Switches in Processor Array Architectures," Proc. of the 1985 Internl. Conf. on Parallel Processing, August 1985.

[33] D.I. Moldovan, "On the Design of Algorithms for VLSI Systolic Arrays," Proc. of IEEE, Vol. 71, No. 1, January 1983.

[34] D.K. Pradhan and J.J. Stiffler, "Error-Correcting Codes and Self-checking Circuits," Computer, Vol. 13, March 1980, pp. 47-54.

[35] D.K. Pradhan and S.M. Reddy, "A Fault-tolerant Distributed Processor Communication Architecture," IEEE Trans. on Computers, Vol. C-31, September 1982, pp. 863-870.

[36] D.K. Pradhan, "Fault-tolerant Architectures for Multiprocessors and VLSI Systems", Proc. of the 13th Symposium on Fault-Tolerant Computing, June 83.

[37] A. L. Rosenberg, "On Designing Fault-Tolerant Arrays of Processors," Tech. report CS-1982-14, Duke University, 1982.

[38] A. L. Rosenberg, "The Diogenes Approach to Testable Fault-Tolerant Arrays of Processors," IEEE Trans. on Computers, Vol. C-32, Oct. 1983, pp. 902-910.

[39] M.R. Samatham and D.K. Pradhan, "A Multiprocessor Network Suitable for Single-Chip VLSI Implementation," Proc. of the 11th Annual Symposium on Computer Architecture, May 1984, pp. 328-337.

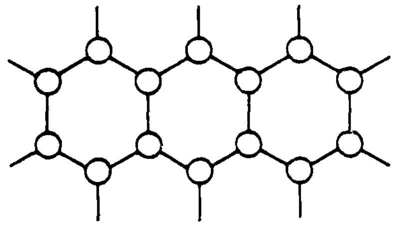
[40] C.H. Sequin and R.M. Fujimoto, "X-Tree and Y-Component," in VLSI Architectures, B. Randell and P.C. Treleaven, eds., Prentice-Hall, 1983, pp.299-326.

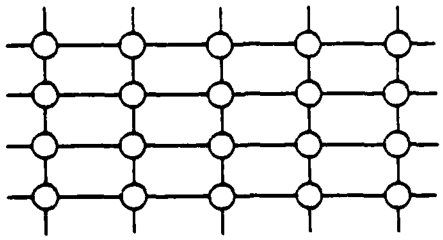
[41] L. Snyder, "Introduction to the Configurable Highly Parallel Computer," Computer, Vol.15, January 1982, pp. 47-56.

[42] C.H. Stapper, A.N. McLaren and M. Dreckman, "Yield Model for Productivity Optimization of VLSI Memory Chips with Redundancy and Partially Good Product," IBM J. Res., Dev., Vol. 24, No. 3, May 1980, pp. 398-409.

[43] C.H. Stapper, F.M. Armstrong and K. Saji, "Integrated Circuit Yield Statistics," Proc. of IEEE, Vol. 71, April 1983, pp. 453-470.

[44] P.C. Treleaven and I.G. Lima, "Japan's Fifth-Generation Computer Systems," Computer, Vol. 15, August 82, pp. 79-88.


(a)


(b)

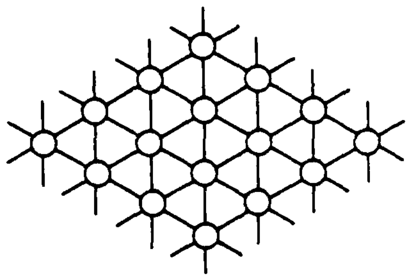
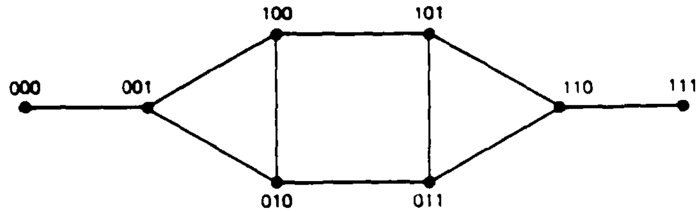
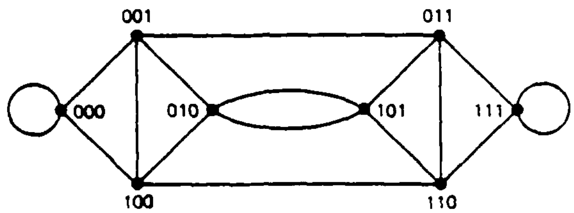

(c)

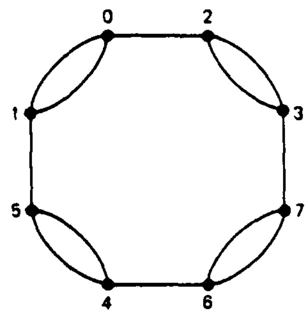
Fig. 1: Mesh connected arrays.



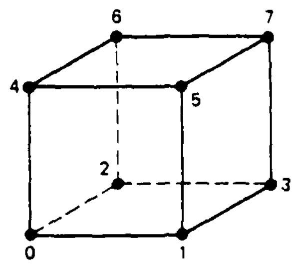
Shuffle-exchange Graph



Shift-and-replace Graph



Cube Connected Cycles



Cube Network

Fig. 2: Algebraic graph networks.

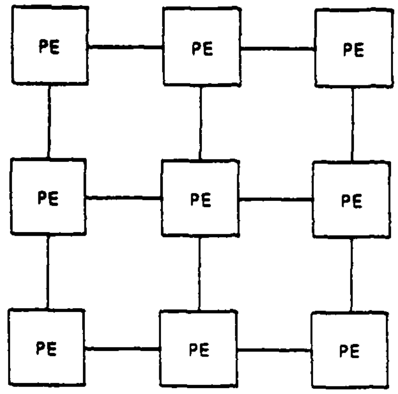


Fig. 3(a): Type 1 architecture using internal switches.

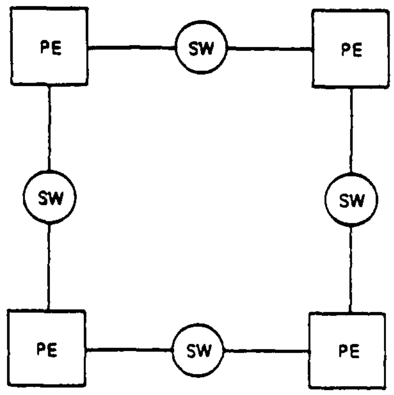


Fig. 3(b): Type 2 architecture using external switches.

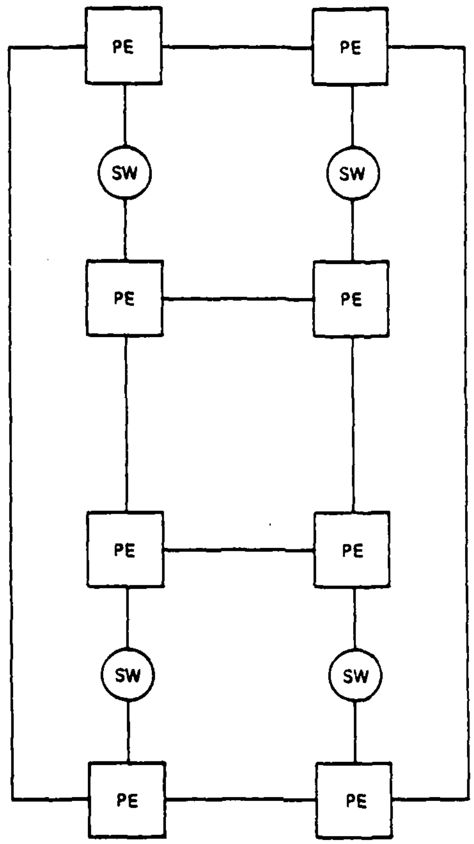


Fig. 3(c): Type 3 architecture.

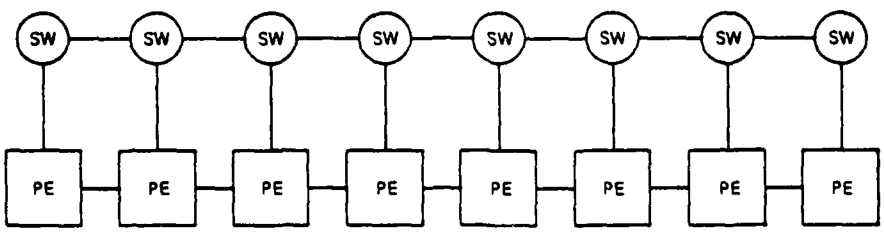


Fig. 3(d): Type 4 architecture.

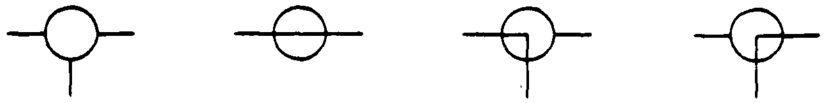


Fig. 4(a): Different switch configurations.

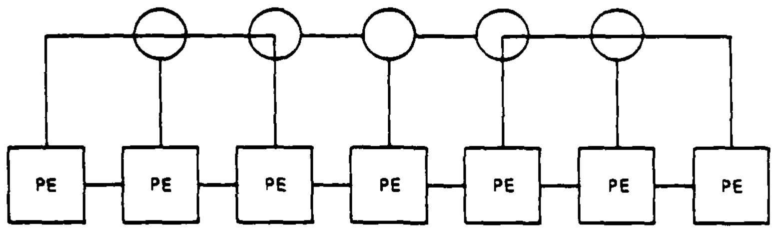


Fig. 4(b): Linear array and binary tree configurations.

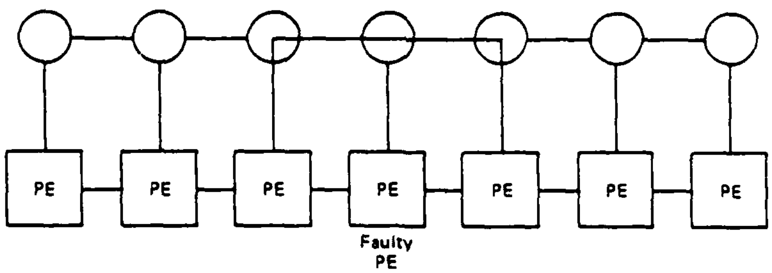


Fig. 4(c): Bypassing faulty PE.

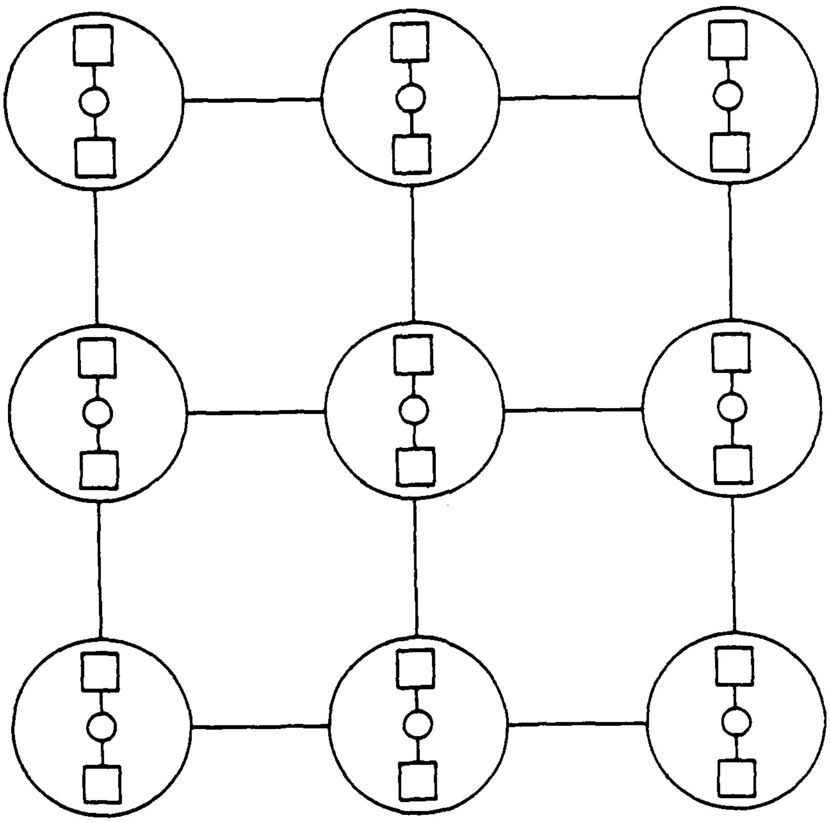


Fig. 5: A 9-node mesh connection.

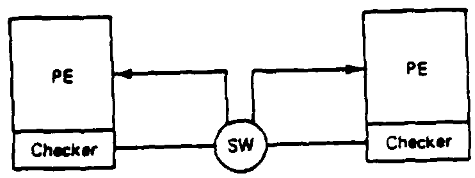


Fig. 6: Fault-tolerant node.

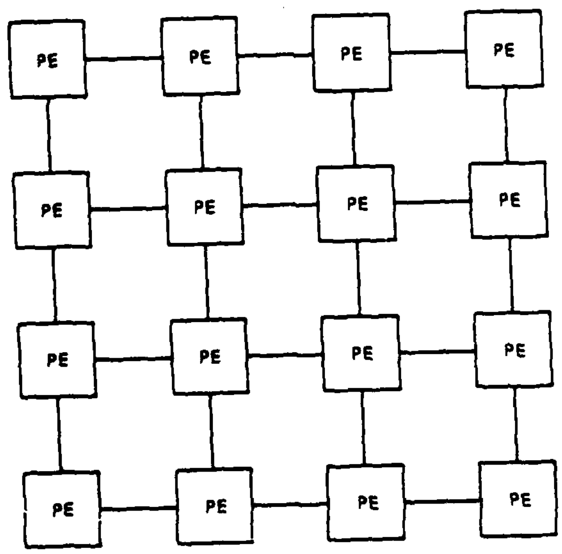


Fig. 7: A 4x4 mesh conr ion.

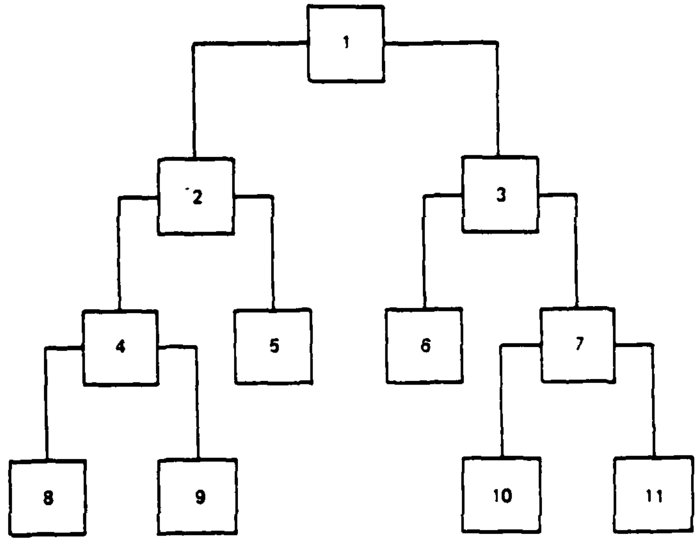


Fig. 8(a): A binary tree configuration.

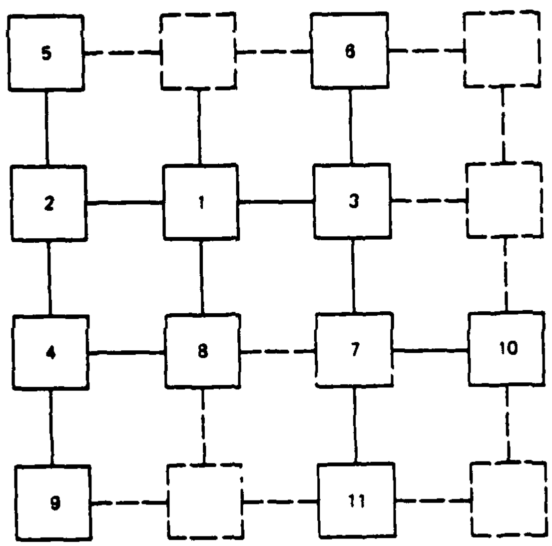


Fig. 8(b): Mapping of the binary tree onto the mesh.

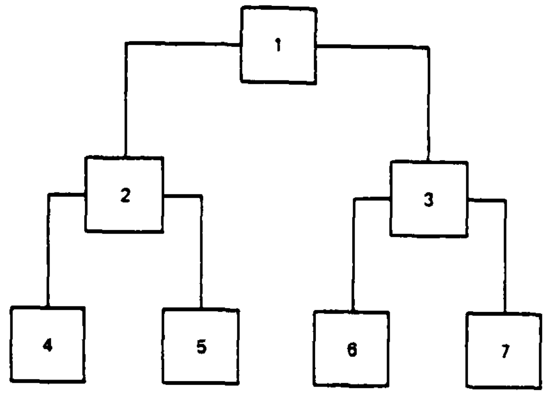


Fig. 9: Reduced binary tree.

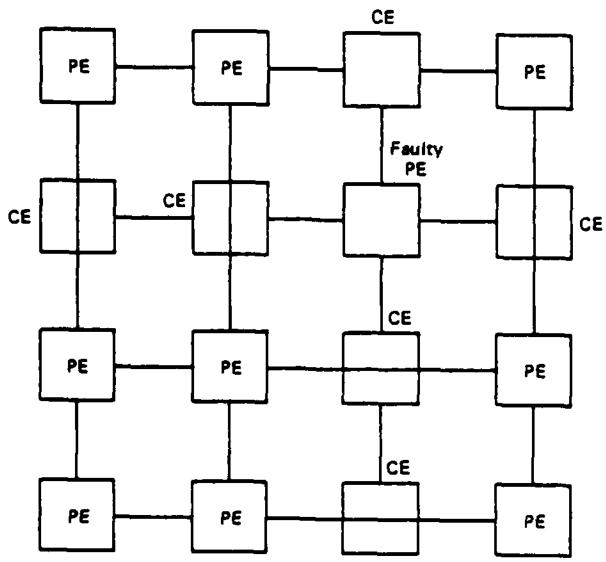


Fig. 10: Reduced 3x3 array.

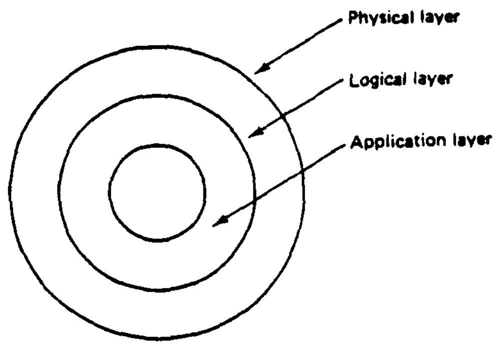


Fig. 11: Topological relationships.

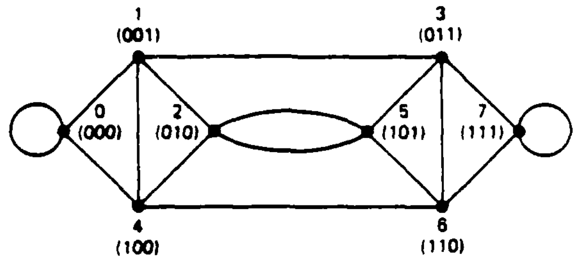


Fig. 12(a): Shift and Replace graph.

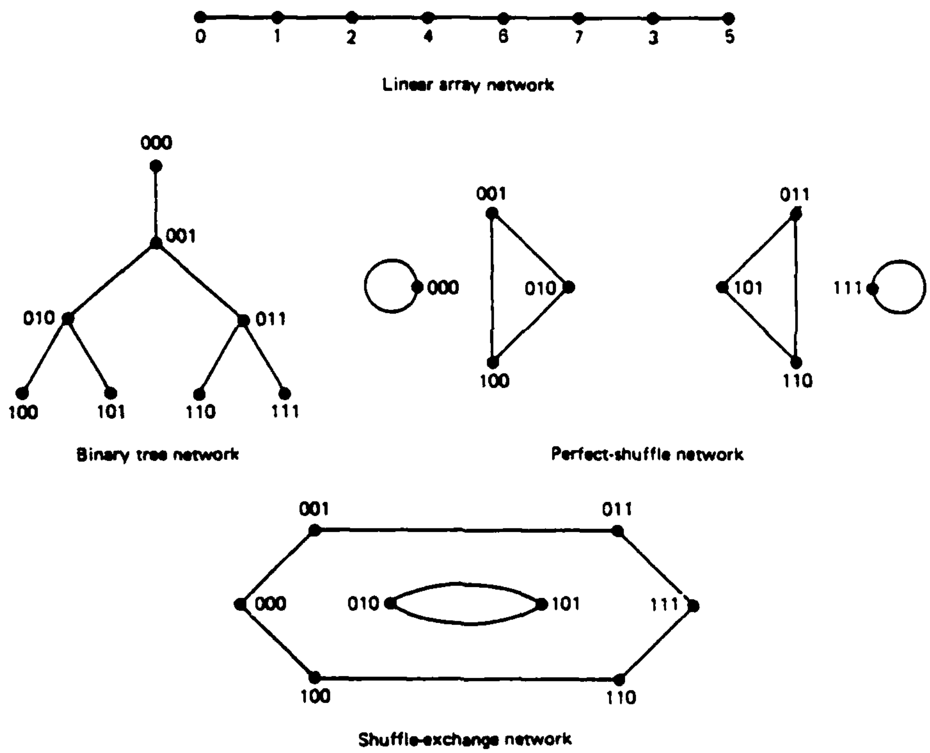


Fig. 12(b): Emulating logical structures on a Shift and Replace graph.

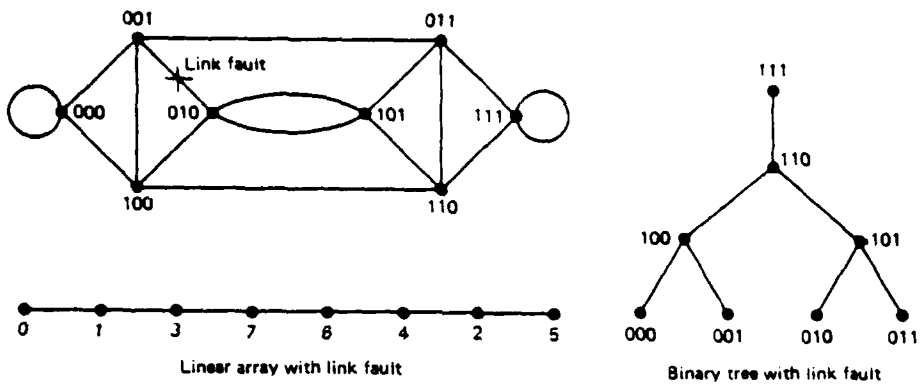


Fig. 13: Emulations in the presence of a faulty link.

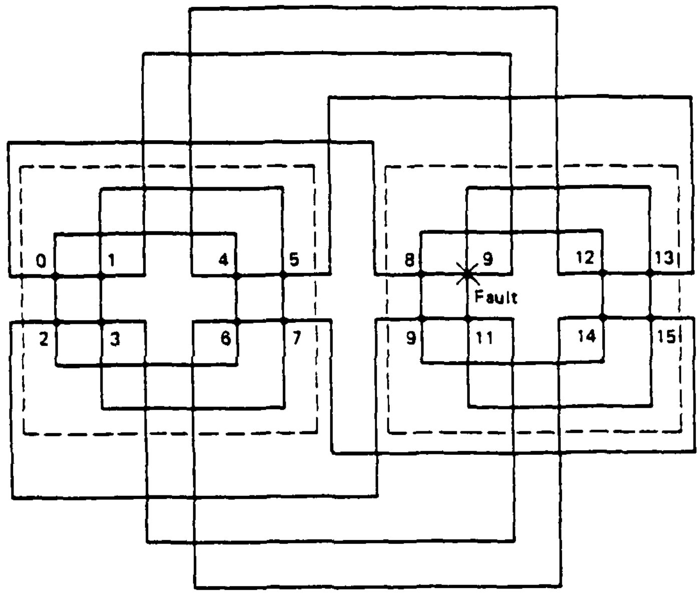


Fig. 14(a): A binary 4-cube partitioned into two 3-cubes with faulty node 9.

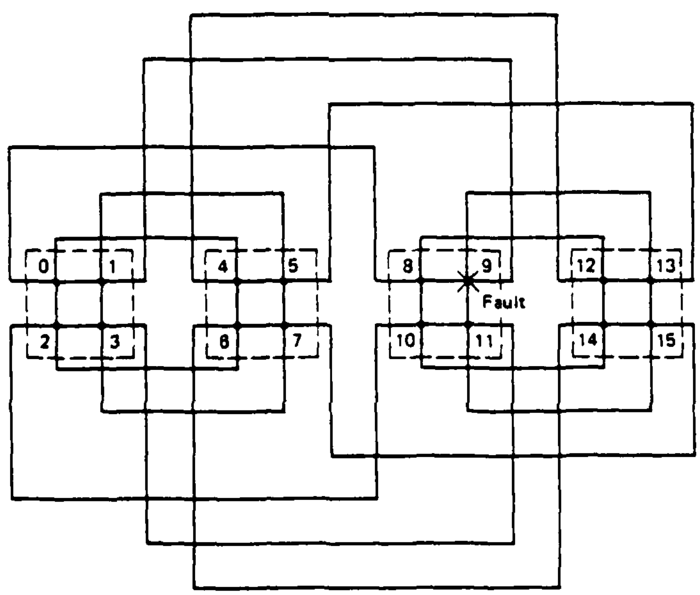
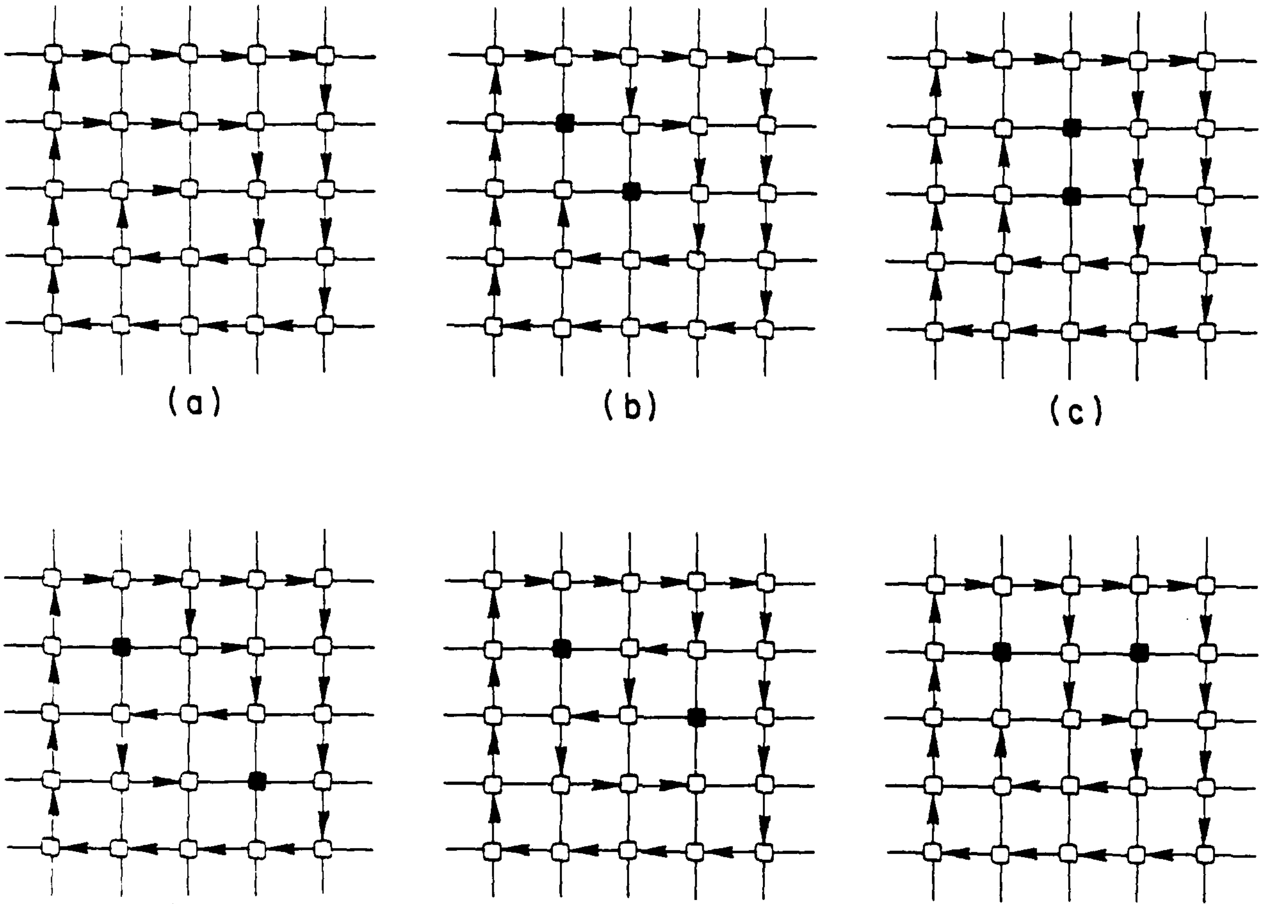


Fig. 14(b): Partitioned binary 4-cube into four 2-cubes with faulty node 9.



(a) (b) (c)

(d) (e) (f)

Fig. 15: Various testing graphs for a 5*5 end-around mesh.

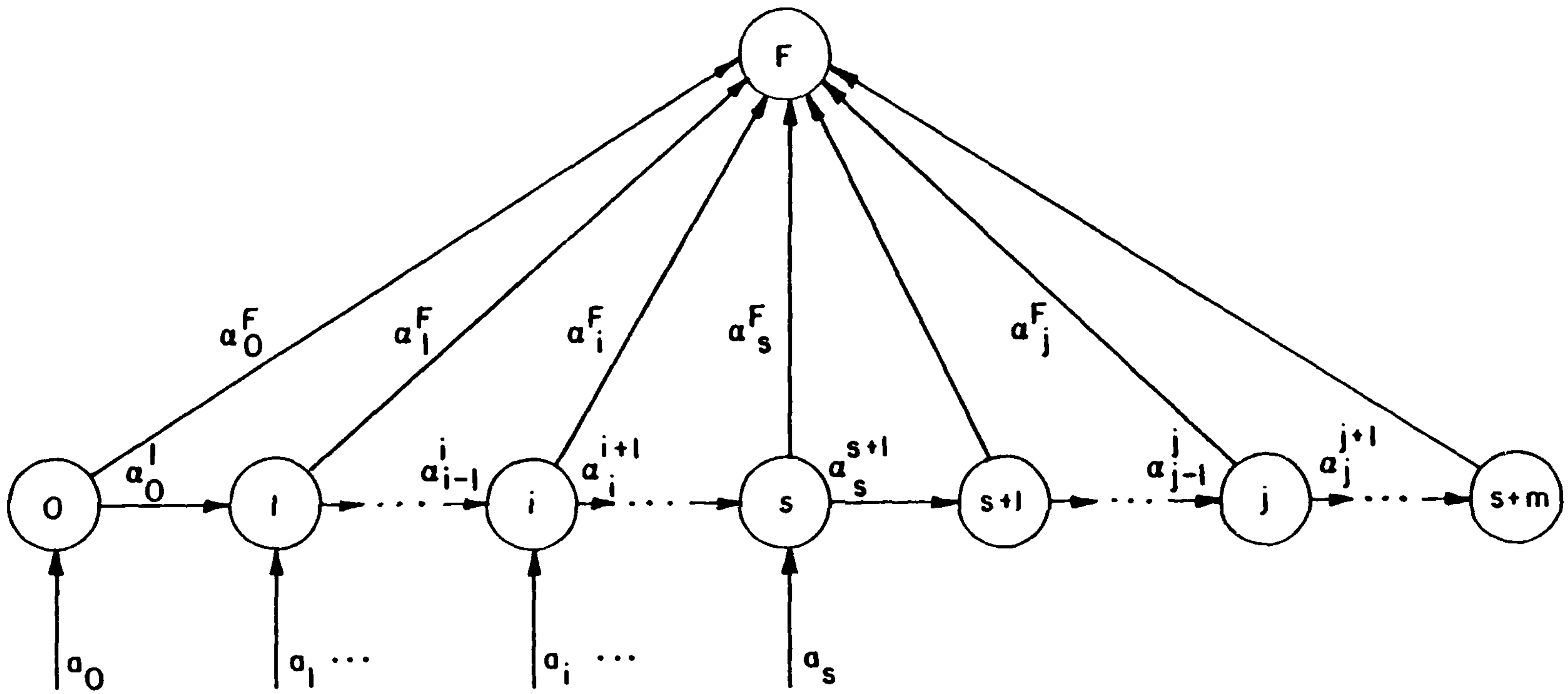


Fig. 16: A Markov model for a multiprocessor with defects and operational faults.

END

DTIC

9-86